Politechnika Świętokrzyska

Wydział Elektrotechniki, Automatyki i Informatyki

**Mateusz Nalepa**

Elektroniczny portfel

Projekt na Technologie Obiektowe

na studiach niestacjonarnych

o kierunku Informatyka

Opiekun projektu:

dr inż. Mariusz Bedla

Kielce, 2019

Spis treści

[1. Przegląd Technologii 3](#_Toc11266776)

[1.1. Język Kotlin 3](#_Toc11266777)

[1.2. Gradle 3](#_Toc11266778)

[1.3. Android Studio 4](#_Toc11266779)

[1.4. SQLite 4](#_Toc11266780)

[1.5. Kodein 4](#_Toc11266781)

[2. Baza danych 5](#_Toc11266782)

[3. Struktura projektu 6](#_Toc11266783)

[3.1. Przykład „activity” jako wyglądu 7](#_Toc11266784)

[3.2. Przykład obsługi „activity” od strony kodu 8](#_Toc11266785)

[4. Projekt aplikacji 9](#_Toc11266786)

[4.1. Ekran główny 9](#_Toc11266787)

[4.2. Ekran dodawania wydatku 10](#_Toc11266788)

[4.3. Ekran edycji 11](#_Toc11266789)

[4.4. Ekran kategorii 12](#_Toc11266790)

[4.5. Ekran historii 13](#_Toc11266791)

[4.6. Ekran średniego wydatku 14](#_Toc11266792)

[5. Podsumowanie i wnioski 15](#_Toc11266793)

[6. Literatura 16](#_Toc11266794)

**Cel projektu**

Celem projektu było stworzenie elektronicznego portfela na platformę Android. Z pomocą aplikacji można kontrolować ilość wydanych pieniędzy w ramach poszczególnych kategorii. Użytkownik może również zarządzać kategoriami w ramach których będzie dodawał nowe wydatki. Może również przeglądać ich historię a także wyświetlić średnią wartość pieniędzy wydaną w ramach danej kategorii w zadanym okresie czasu.

# przegląd Technologii

W ramach projektu zostały wykorzystane następujące technologie i języki programowania obiektowego.

## Język Kotlin

Statycznie typowany język programowania który działa na wirtualnej maszynie Javy. Rozwijany głównie przez programistów z JetBrains, twórców wielu środowisk programistycznych, między innymi IntelliJ. Język ten jest w pełni interoperacyjny z Javą, dzięki czemu cześć projektu może być napisana w Javie, a inna część w Kotlinie. Praktycznie najważniejszą zaletą tego języka jest fakt, iż jest on „null safety”, oznacza to, że zmienna która odnosi się do instancji obiektu nie może być wartością null, o ile programista wyraźnie tego nie zaznaczy poprzez dodanie znaku pytajnika „?” prze deklaracji typu zmiennej.

**val age**: Int? = **null**

Inną ważną rzeczą jest to, że język ten wykorzystuje wnioskowanie typu, oznacza to, że podczas tworzenia obiektu zbędne jest podanie jego typu, gdyż kompilator będzie wiedział, jaki jest typ zmiennej.

**val name** = **"Hodor"**

Na podstawie napisu kompilator sam wywnioskuje, że dany typ zmiennej to String.

Język ten został wybrany do projektu ze względu na to, że od Maja 2017 to właśnie Kotlin jest oficjalnym językiem programowania na platformę Android.

## Gradle

Narzędzie służące do zarządzania cyklem życia a także budowaniem projektu. Z jego pomocą można bardzo łatwo dodawać nowe zależności do projektu a także skonfigurować repozytorium, w którym będą przechowywane poszczególne wersje aplikacji. Dodanie nowej zależności ogranicza się praktycznie do jednej linijki której przykład wygląda następująco:

compile **"com.github.salomonbrys.kodein:kodein:**$kodein**"**

Jego główną zaletą w porównaniu do innych narzędzi takich jak np. Maven jest to, że wykorzystuje on język Groovy, dzięki czemu jest bardziej elastyczny. Dlatego też firma Google wykorzystuje Gradle’a jako oficjalnie narzędzie do zarządzania projektami napisanymi na platformę Android.

## Android Studio

Środowisko programistyczne bazujące na IntelliJ. Jest zalecanym środowiskiem programistycznym do pisania aplikacji na platformę Android. Oferuje on wiele możliwości, takich jak autouzupełnianie kodu, wykrywanie duplikatów, podgląd wyglądu aplikacji zanim jeszcze zostanie uruchomiona na rzeczywistym urządzeniu. Pozwala również na debugowanie aplikacji zainstalowanej na fizycznym urządzeniu bądź na emulatorze. Oferuje również przeładowanie fragmentów kodu na uruchomionej aplikacji, tzw „hot swap”. Oznacza to, że w trakcie działania programu można zmienić jego implementację. Środowisko to wspiera również praktycznie wszystkie systemy kontroli wersji z jakich może zechcieć skorzystać użytkownik, takich jak: SVN, GIT.

## SQLite

Mimo ogromnego rozwoju urządzeń elektrotechnicznych jak i również samych języków programowania, to jednak urządzenia mobilne oferują znacznie mniej, niż jednostki centralne. Jednakże potrzeba zapisywania danych jest wszędzie, dlatego też platforma Android wykorzystuje system zarządzania bazą danych zwany **SQLite**. Oznacza to po prostu tyle, że jest to lekka wersja pełnych systemów zarządzania bazami danych. Cała baza znajduje się w jednym pliku więc dostęp do tej bazy jest zdecydowanie prostszy, gdyż nie udostępnia ona opcji autoryzacji użytkowników. Nie wspiera ona również takiego typu, jak boolean.

## Kodein

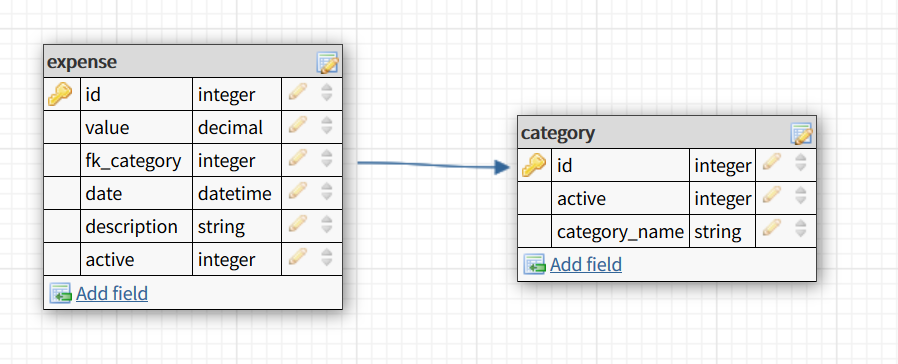
Lekki framework który umożliwia obsługę wstrzykiwania zależności. Programista musi sam deklarować wszystkie komponenty aplikacji, lecz pisania i utrzymanie projektu jest wtedy zdecydowanie prostsze.

# Baza danych

W ramach stworzenia elektronicznego portfela została stworzona baza jedynie z dwóch tabel połączonych relacją jeden do wielu. Stworzone tabele przechowuję następujące dane:

* „category” – przechowuje informacje o kategoriach wykorzystanych w projekcie
* „expense” – przechwouje informacje na temat danego wydatku, takie jak data, koszt, opis oraz wskazuje również na kategorię danego zakupu.

Schemat stworzonej bazy wygląda następująco:



Rysunek 1 Schemat bazy danych

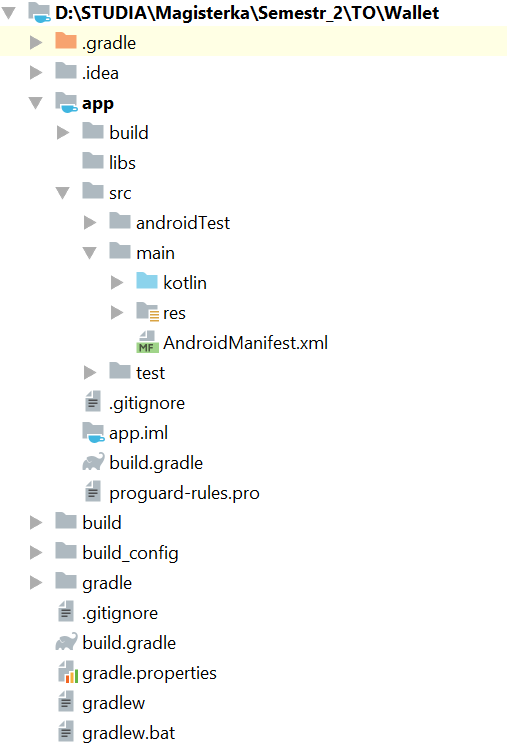
Przy pierwszym uruchomieniu aplikacji baza jest zasilana domyślną listą kategorii w której skład wchodzą między innymi: ”Zakupy”, „Mieszkanie” oraz „Leki”.

Baza danych została wygenerowana na zapytania SQL który w projekcie wygląda następująco:

**object** CategoryEntry {  
 **const val TABLE\_NAME** = **"category"  
 const val ID** = **"id"  
 const val COLUMN\_CATEGORY** = **"category\_name"  
 const val COLUMN\_ACTIVE** = **"active"**}  
  
**object** ExpenseEntry {  
 **const val TABLE\_NAME** = **"expense"  
 const val ID** = **"id"  
 const val COLUMN\_AMOUNT** = **"value"  
 const val COLUMN\_CATEGORY\_ID** = **"fk\_category"  
 const val COLUMN\_DATE** = **"date"  
 const val COLUMN\_DESCRIPTION** = **"description"  
 const val COLUMN\_ACTIVE** = **"active"**}  
  
  
  
**object** DatabaseSchema {  
  
 **const val SQL\_CREATE\_CATEGORY\_TABLE** =  
 **" CREATE TABLE IF NOT EXISTS ${**CategoryEntry.**TABLE\_NAME} ("** +  
 **"${**CategoryEntry.**ID} INTEGER PRIMARY KEY,"** +  
 **"${**CategoryEntry.**COLUMN\_ACTIVE} INTEGER,"** +  
 **"${**CategoryEntry.**COLUMN\_CATEGORY} TEXT UNIQUE); "  
  
 const val SQL\_CREATE\_EXPENSE\_TABLE** =  
 **" CREATE TABLE IF NOT EXISTS ${**ExpenseEntry.**TABLE\_NAME} ("** +  
 **"${**ExpenseEntry.**ID} INTEGER PRIMARY KEY,"** +  
 **"${**ExpenseEntry.**COLUMN\_AMOUNT} NUMERIC,"** +  
 **"${**ExpenseEntry.**COLUMN\_ACTIVE} INTEGER,"** +  
 **"${**ExpenseEntry.**COLUMN\_DESCRIPTION} TEXT,"** +  
 **"${**ExpenseEntry.**COLUMN\_CATEGORY\_ID} INTEGER,"** +  
 **"${**ExpenseEntry.**COLUMN\_DATE} DATETIME,"** +  
 **" FOREIGN KEY(${**ExpenseEntry.**COLUMN\_CATEGORY\_ID}) "** +  
 **" REFERENCES ${**CategoryEntry.**TABLE\_NAME}(${**CategoryEntry.**ID})); "**}

# Struktura projektu

Projekt został napisany w środowisku programistycznym Android Studio. Struktura projektu wygląda następująco:



Rysunek 2 Struktura projektu

* build.gradle - znajdujący się na najwyższym poziomie hierarchii katalogów zawiera ogólną konfigurację projektu, to znaczy fakt tego, iż projekt wykorzystuję język Kotlin.
* build.gradle - zagnieżdżony w module „app” zawiera listę wszystkich zależności wykorzystanych w projekcie.
* AndroidManifest.xml – zawiera nazwę projektu jak i listę wszystkich „activity”, to znaczy listę wszystkich ekranów które obsługuje aplikacja.

W platformie Android każda „strona” jaką może wyświetlić użytkownik jest nazywana aktywnością. Każda aktywność składa się z dwóch komponentów, takich jak:

* xml – plik w postaci xml który opisuje wygląd danego ekranu
* klasa activity – klasa, które rozszerza klasę bazową AppCompatActivity odpowiedzialną za wyświetlania ekranu użytkownikowi

W niniejszym rozdziale jest zaprezentowany przykład activity dla strony głównej aplikacji.

## Przykład „activity” jako wyglądu

Layout, czyli wygląd jest następujący:

* *<?***xml version="1.0" encoding="utf-8"***?>*<**android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"  
   xmlns:app="http://schemas.android.com/apk/res-auto"  
   xmlns:tools="http://schemas.android.com/tools"  
   android:id="@+id/drawerLayout"  
   android:layout\_width="match\_parent"  
   android:layout\_height="match\_parent"  
   tools:context=".MainActivity"**>  
    
   <**android.support.design.widget.AppBarLayout  
   android:layout\_width="match\_parent"  
   android:layout\_height="wrap\_content"  
   android:theme="@style/AppTheme.AppBarOverlay"**>  
    
   <**android.support.v7.widget.Toolbar  
   android:id="@+id/toolbar"  
   android:layout\_width="match\_parent"  
   android:layout\_height="?attr/actionBarSize"  
   android:background="?attr/colorPrimary"  
   app:popupTheme="@style/AppTheme.PopupOverlay"** />  
   <**include layout="@layout/menu"**></**include**>  
    
   </**android.support.design.widget.AppBarLayout**>  
    
  </**android.support.v4.widget.DrawerLayout**>
* DrawerLayout – layout, który jest nadrzędny dla wszystkich ekranów aplikacji, z jego pomocą jest wyświetlany pasek górny aplikacji. Jego definicja znajduje się w elemencie „AppBarLAyout”
* Include – element który pozwala na załączenie fragmentu innego layoutu na innym, dzięki temu unika się powielania pisania identycznych fragmentów wyglądu aplikacji
* Każdy layout posiada unikalne id, w tym przypadku jest to „drawerLayout”.

## Przykład obsługi „activity” od strony kodu

Obsługa activity wygląda następująco:

**class** MainActivity : AppCompatActivity(), AppCompatActivityInjector {  
  
 **override val** injector: KodeinInjector = KodeinInjector()  
 **private lateinit var** drawerLayout: DrawerLayout  
 **private lateinit var** toggle: ActionBarDrawerToggle  
  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_main)  
 initMainActivity()  
 initializeInjector()  
 }  
  
 @SuppressLint(**"RestrictedApi"**)  
 **private fun** initMainActivity() {  
 drawerLayout = findViewById(R.id.drawerLayout)  
 toggle = ActionBarDrawerToggle(**this**, drawerLayout, R.string.open, R.string.close)  
 drawerLayout.addDrawerListener(toggle)  
 toggle.syncState()  
 supportActionBar?.setDefaultDisplayHomeAsUpEnabled(**true**)  
 setSupportActionBar(toolbar)  
 }  
  
 **fun** showAddExpenseActivity(view: View) {  
 **val** intent = Intent(**this**, AddExpenseActivity::**class**.java)  
 startActivity(intent)  
 }

**fun** showHistoryActivity(view: View) {  
 **val** intent = Intent(**this**, ExpenseHistoryActivity::**class**.java)  
 startActivity(intent)  
 }

**override fun** onDestroy() {  
 destroyInjector()  
 **super**.onDestroy()  
 }  
}

W powyższym pliku najbardziej kluczowe informacje to:

* rozszerzenie klasy „AppCompatAcvitity”, dzięki temu klasa „MainActivity” jest świadoma tego, że będzie wyświetlać layout użytkownikowi
* metoda „onCreate” – metoda którą trzeba nadpisać w celu wyświetlenia danego layoutu, w celu wyświetlenia konkretnego layoutu należy podać jego id w metodzie „setContentView”
* Kodein – wszystko co ma w nazwie „kodein” odpowiada za to, że do danego activity można wstrzykiwać zależności

# Projekt aplikacji

Bardzo ważnym elementem każdej aplikacji jest zaprojektowanie czytelnego i przejrzystego interfejsu graficznego. Z jego pomocą użytkownik może łatwo nawigować po aplikacji bez zastanawiania się nad tym, w jaki sposób uzyskać dostęp do danej funkcjonalności.

## Ekran główny

Ekran główny jest to główny ekran, jaki zobaczy użytkownik na starcie aplikacji. Na tym ekranie użytkownik widzi nazwę aplikacji jak i również może przejść do każdego innego ekranu.



Rysunek 3 Główny ekran aplikacji

## Ekran dodawania wydatku

W ramach tego ekranu można dodać nowy wydatek do bazy, domyślną kategorią są zawsze „Zakupy”. Data wydatku ustawia się automatycznie na aktualną datę, jednakże można ją zmienić na inną. Kwota podlega walidacji. Po dodaniu wydatku użytkownik jest automatycznie przekierowany na stronę z historią wydatków i widzi odpowiedni komunikat.



Rysunek 4 Ekran dodawania wydatku

## Ekran edycji

Ekran edycji jest praktycznie identyczny do ekranu wydatku aplikacji, z tą różnica, że może być on aktywowany jedynie poprzez ekran „Historia”.



Rysunek 5 Ekran edycji wydatku

## Ekran kategorii

Z pomocą tego ekranu użytkownik może wyświetlać wszystkie kategorie znajdujące się w danej aplikacji. Na początku otrzymuje on listę domyślnych kategorii które może usunąć. Usunięcie danej kategorii oznacza również usunięcie wszystkich wydatków związanych z daną kategorią.



Rysunek 6 Ekran kategorii

## Ekran historii

Na ekranie historii użytkownik może przeglądać wszystkie wydatki jakich dokonał w kolejności od najwcześniejszego do najstarszego. Użytkownik może wybrać zakres dat jak i również kategorię. Domyślnie są wyświetlane wszystkie wydatki. Po rozwinięciu poszczególnego wydatku użytkownik widzi jego szczegóły a także może edytować bądź usunąć dany wydatek.



Rysunek 7 Ekran historii

## Ekran średniego wydatku

Na tym ekranie użytkownik może zobaczyć jaką średnią wartość pieniędzy wydaje w ciągu dnia w zadanym okresie czasu z podziałem na poszczególną kategorię. Domyślnie jest wyświetlaną wartość jaką użytkownik średnio wydał w ciągu ostatnich 7 dni.



Rysunek 8 Ekran średniego wydatku

# Podsumowanie i wnioski

Największą trudnością w stworzeniu elektronicznego portfela było stworzenie pierwszego Layoutu oraz połączenia do bazy. Jednakże po stworzeniu pierwszego Layoutu stworzenie pozostałych nie było już tak wielkim wyzwaniem.

Aplikacja spełnia wszystkie podstawowe wymagania, w odczucia autora najważniejszą z nich jest wyświetlenie średniej wartości pieniędzy wyświetlonej w danej kategorii w zdanym okresie czasu, gdyż żadna z podobnych aplikacji dostępnych na „Google Play” nie oferuje takiej funkcjonalności.

Interfejs aplikacji jest bardzo przejrzysty i czytelny, więc absolutnie nikt nie powinien mieć problemu z jego obsługą. Wszystkie jego elementy są przejrzyste i czytelne.

Jednym z kolejnych momentów rozwoju aplikacji było by dodanie bocznego wysuwanego menu jak i również integracja aplikacji z aplikacją mobilną banku, w celu automatycznego dodawania wydatków dokonanych przez aplikację mobilną banku, lecz ze względów bezpieczeństwa, to raczej było by bardzo trudne do zrealizowania.

# Literatura

1. <https://kotlinlang.org/>
2. <https://gradle.org/>
3. <https://developer.android.com/studio>
4. <https://github.com/Kodein-Framework/Kodein-DI/>
5. <https://www.sqlite.org/index.html>
6. <https://developer.android.com/guide>